Trust Factors and Insider Threats in Permissioned Distributed Ledgers

An analytical study and evaluation of popular DLT frameworks

Benedikt Putz and Günther Pernul

Department of Information Systems, University of Regensburg, Regensburg, Germany
{benedikt.putz, guenther.pernul}@ur.de

Abstract. Permissioned distributed ledgers have recently captured the attention of organizations looking to improve efficiency, transparency and auditability in value network operations. Often the technology is regarded as trustless or trust-free, resulting in a false sense of security. In this work, we review the various trust factors present in distributed ledger systems. We analyze the different groups of trust actors and their trust relationships to the software layers and inherent components of distributed ledgers. Based on these analyses, we investigate how insiders may conduct attacks based on trust in distributed ledger components. To verify practical feasiblity of these attack vectors, we conduct a technical study with four popular permissioned distributed ledger frameworks: Hyperledger Fabric, Hyperledger Sawtooth, Ethereum and R3 Corda. Finally, we highlight options for mitigation of these threats.

Keywords: trust frameworks · distributed systems security · distributed ledger technology · insider threat

1 Introduction

Distributed ledger technology (DLT) offers great potential to decentralize operations in collaborative business networks and may even enable new business models [46]. Benefits include cost reduction and increased transparency in information sharing between organizations. However, great potential also entails great risks and potential security issues. Recent reviews regarding the future of blockchain technology have pointed out the need to study security and trust aspects of DLT [13, 51].

Blockchain and DLT are often described as trustless or trust-free alternatives to currently established centralized systems (see [29, 30, 36]). In this work, we take a closer look at the usage of permissioned distributed ledgers and examine whether it can really be considered "trustless". The term "trustless" originates from the decentralization of control in distributed ledger networks [29], which aims to replace trusted third parties. The goal of this work is to establish a framework for reasoning about trust elements in permissioned distributed

ledgers. These trust elements can also be exploited by insiders, who are aware of them and in control of crucial components of the trust system.

Insider threats are a tough cybersecurity problem, which remains difficult to detect and prevent due to abuse of legitimate access permissions by the attacker. According to the roadmap of cybersecurity research by the US department of Homeland Security, insider threats are one of the "hard problems" of information security research [50]. Similarly, the European cybersecurity agency ENISA's threat landscape report lists insider threats among the top 10 information security threats, with 77% of companies' data breaches caused by insiders [23].

Insider threats are particularly relevant for distributed ledgers operated by a network of independent organizations. These networks are called *permissioned*, since they are operated by a restricted set of authenticated member nodes. In this scenario, intra-organizational insiders are supplemented with external insiders [25] from other organizations, who also have access to information shared on the network. According to a recent survey on enterprise adoption of DLT, there are at least 50 corporations with valuations larger than $1 billion looking to implement DLT to trade digital assets [11]. Many of these are financial institutions looking to trade high-value assets, leading to an attractive target for insider attacks.

To appropriately assess trust in distributed ledgers, our trust definition is based on software trust as defined by Amoroso and Watson [3]: *"Software trust is the degree of confidence that the software will be acceptable for 'one's needs'. It is established after one has become convinced, presumably based on a meaningful set of information, that the software does not include flaws that will prevent it from meeting its requirements."*

Besides trust in software components, the second form of trust is related to assessments of the human agents that collectively control the distributed system (hereafter referred to as trust actors). We follow Gambetta's definition of trust [28]: *"Trust (or, symmetrically, distrust) is (...) the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (...) and in a context in which it affects his own action."*

The remainder of this work is structured as follows: In Section 2 we give a short overview of related work concerning trust and insider threats with regard to distributed ledgers. Subsequently, we analyze trust actors, layers and components of permissioned distributed ledgers in Section 3. Based on our assessment of trust factors we identify relevant attack vectors for the different groups of insiders in Section 4. In Section 5 we perform a threat analysis for 5 popular distributed ledger frameworks, examining how insiders might exploit these vectors in practice. Finally, we wrap up by giving recommendations for future research in Section 6 and summarize our results in Section 7.

2 Related work

While research on trust in blockchain systems is still scarce, the Global Blockchain Benchmarking Study by the University of Cambridge points out

that blockchains always require some degree of trust [32]. Recent blog posts have highlighted trust factors in public permissionless blockchains [47]. Permissioned blockchains rely on similar trust primitives: trust in application code, network/cryptographic protocols and hardware. We aim to expand upon these notions by exploring the trust factors in more detail.

Overall, only partial aspects of trust in blockchain networks have been studied. Locher et al. [38] create a formal model to examine whether a distributed ledger may fully replace a trusted third party. In the process, they also evaluate previously proposed use cases of DLT that still require trust in other organizations and third parties. Hawlitschek et al. [30] review the conceptualization of trust in the blockchain environment. They argue that it is difficult to assess whether a system is actually trust-free or not. Correspondingly, another study claims that blockchain shifts trust from central authorities towards algorithms [39]. However, for this shift to be successful, the algorithms need to be trusted. Smart contracts represent the application-level algorithms, and their control flow immutability and independence of third parties have been shown to be lacking [26]. In addition to algorithmic properties, researchers have also studied user trust of different stakeholder groups from an HCI perspective [45]. In summary, research has established the existence of trust factors that contradict the claim of a trust-free system [30], but a comprehensive model of trust relationships is still missing.

Despite the severity of insider threats as pointed out by government agency assessments [23], research regarding insider threats in distributed ledger consortia is still scarce. Numerous surveys on the security of blockchain systems have been carried out [17,35], but none of them have focused on insiders in particular. We intend to fill this research gap and provide direction for future research.

In particular, we go beyond existing work by presenting a novel model of trust actors, their relationships and trusted DLT software components. We use this model to derive insider threats that organizations face when implementing permissioned DLT. By analyzing frameworks popular in both industry and research, we show that these attacks are applicable in practice. To protect against these threats, we outline technical and organizational options to mitigate the insider threats at hand.

3 Trust factors in permissioned distributed ledgers

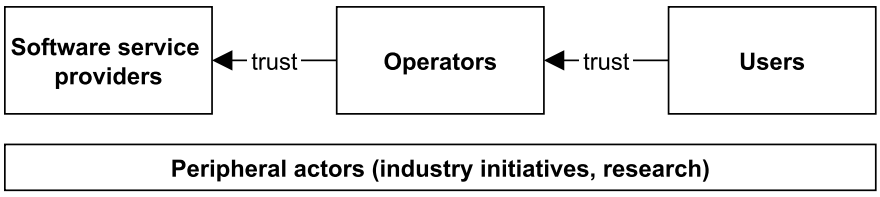

Fig. 1. Trust actors and relationships.

As noted by other researchers, the requirement for trust does not disappear simply by employing a distributed ledger. Instead, trust shifts from trust in other organizations to trust in the technology and its operation [38]. In this section we focus on analyzing the different components of a distributed ledger system to establish trust actors, layers and components.

Before examining the trust factors in detail, trust actors need to be identified. There are four types of actors in the DLT ecosystem, three of which directly contribute to trust relationships in a consortium: software service providers, operators and users [32]. Peripheral actors (i.e. industry initiatives and researchers) do not directly interact with consortium networks, as they are not involved with building or operating DLT software. Nevertheless, they contribute by developing standards, methods and paradigms to solve current technical challenges [32].

An overview of the resulting trust hierarchy is shown in Figure 1. **Software service providers** (SSPs) develop the software components of a distributed ledger consortium. They are trustees responsible for creating trust in the technology by developing secure and reliable applications. **Operators** represent distinct groups of actors responsible for running the distributed ledger overlay network and applications built on top of it. They act as both trustors and trustees: they trust the chosen DLT software to operate as expected and are also trusted by their users to provide reliable operations. Finally, **Users** place their trust in these applications and rely on them to work as advertised, without knowledge of the lower layers.

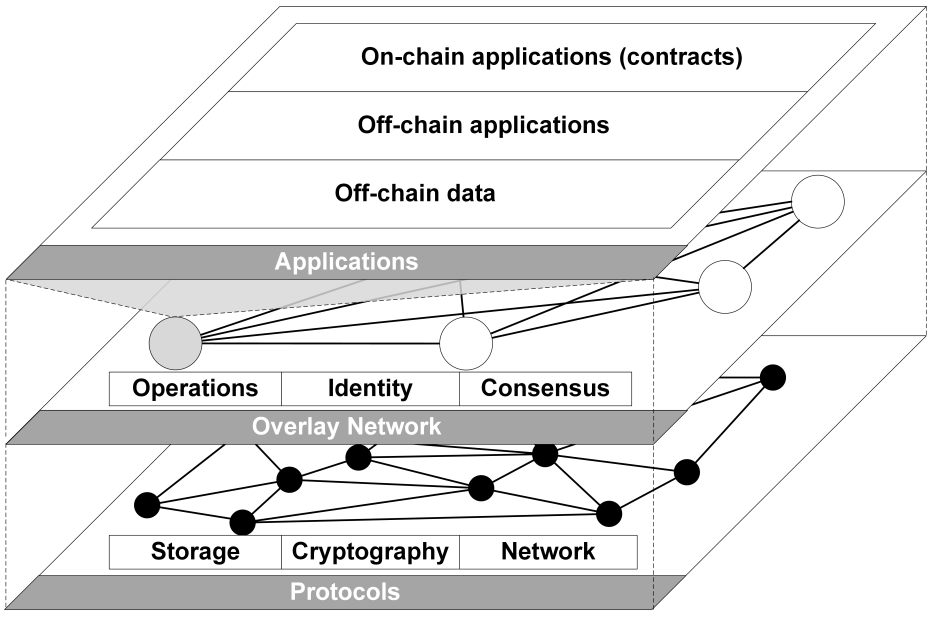

Fig. 2. Distributed ledger software layers and components.

The trust actors in Figure 1 interact using a permissioned distributed ledger network, which consists of several software layers. Figure 2 shows the layers and the software components on each layer. They are derived from the three-layer view of Component Based Systems: platform, middleware and application [43]. The platform in this case consists of various underlying *protocols* responsible for storage, cryptography and network communication. Also part of the platform, but out of scope for this work, are the operating system and hardware layers. The middleware is represented by an *overlay network*, which provides configurable functionality for operations, identity management and distributed consensus. The *applications* layer provides replicated application logic (on-chain) and external logic and data (off-chain). These off-chain applications integrate with the framework by reading/writing data through its APIs. Each of the components within these layers requires software trust: it should be working correctly and not be maliciously exploitable. Since layers are built on top of each other, software bugs or vulnerabilities may propagate upwards to affect higher layers. In the following subsections we elaborate in detail on the layers' components and how they are involved in creating trust in the distributed ledger.

Figure 3 integrates trust actors from Figure 1 with the layers from Figure 2 by illustrating which trust actors govern each system layer. While software service providers are involved in the development of all three layers, operators do not interact with the underlying protocols. They merely configure the network framework and develop applications on top of it. Meanwhile, users only interact directly with the application layer.

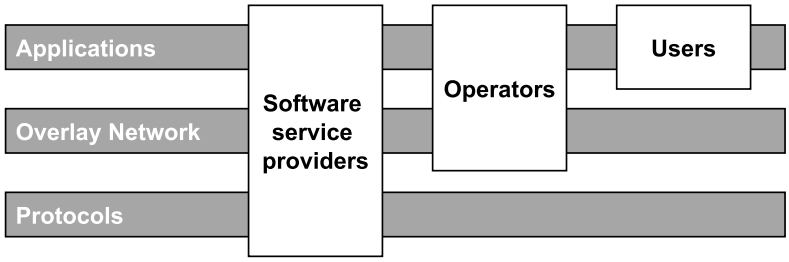

Fig. 3. Intersection of trust actors and layers.

3.1 Protocols

Storage, cryptographic and network protocols carry out the low-level tasks instructed by the distributed ledger framework and its applications. For this reason, they form the trust basis of the network.

Storage. A key property of blockchain-based systems is the goal of maintaining immutability of the underlying chain of blocks. Transaction and block

metadata are stored in relational or key-value databases locally on each node. Replication integrity is assured through distributed consensus. The claimed immutability is a key factor in enabling trust in the technology, but it only holds if storage and network protocols can be trusted.

Cryptography. A manifold of cryptographic protocols are involved in distributed ledger operation. They include:

- hash functions for integrity assurance (hash chains, Merkle trees and proofs)
- public key cryptography (authentication of consensus protocol messages and user-submitted transactions)
- zero knowledge proofs (privacy-preserving transactions)
- symmetric encryption (on-chain confidentiality)

All of these protocols are trusted to not have design or implementation flaws. Many frameworks assemble their cryptography from a variety of sources, including standard libraries, external libraries and custom implementations (see Section 5). While some developers such as the Hyperledger open-source project perform third-party security audits [33], even the most diligent audits may miss vulnerabilities. Operators and users must trust protocol design and implementation, often without the ability to verify due to lack of cryptographic expertise.

Network protocols. A fundamental trust factor for distributed ledger node communication is untampered operation of the underlying network. Distributed ledger networks are overlay networks, so they rely on P2P routing algorithms and message dissemination protocols for communication. Since all peers are equal, any single peer may cause disruption in the network by sending anomalous or malicious traffic. This may cause unexpected behavior and violate the aforementioned trust assumption.

3.2 Overlay network

A distributed ledger network is a permissioned overlay network that consensually maintains a replicated ledger. In this overlay network, independent operators deploy a software framework previously agreed upon (i.e. Hyperledger Fabric). There are several tasks that each operator is trusted to fulfill by other participants: carry out operations tasks, maintain identity and access privileges and participate in consensus. The network layer also provides virtualization capabilities for replicated deterministic application execution in the application layer.

Operations. These independent organizations trust each other to perform node setup and operation without malicious manipulation. While there is some fault tolerance built into distributed ledgers (see Section 3.2), more than two-thirds of operators must behave honestly if byzantine-fault tolerant protocols are used.

Operation includes consensual admission/removal of members, on-chain application upgrades and framework upgrades. For the latter, all operators must agree on a coordinated time for system maintenance. Provided that all partners agree on the schedule, some partners might not be able to upgrade successfully [20]. Even if the failure to upgrade is not of malicious intent, it might

pose considerable challenges to all involved parties, like setting up a new network and migrating data. Overall, the process requires significant trust in other organizations that cannot be mitigated by technology.

Identity and access control. Since the network is made up of independent organizations, each entity must be able to manage its users independently. This means that every organization must trust the others to properly manage identities and access rights. Since internal screening and job rotation processes are usually opaque to others in collaborative business networks [25], this can be considered blind trust.

Fundamental to permissioned distributed ledgers is an access control mechanism that ensures only authorized operators are part of the network. This is usually realized by assigning each node a public key, which is known to the other nodes and used to authenticate and secure communication. While admission/removal of node operators is based on majority consensus, other participants must be trusted to only accept legitimate new members. Additionally, compromise of a single node's credentials may undermine the trust assumption of a closed network.

Consensus. The consensus protocol is the distributed agreement protocol at the core of a permissioned distributed ledger, allowing all nodes to share a single replicated state. However, due to fundamental limitations underlying deterministic replicated state machines, less than one-third of participants may be malicious at the same time [12]. This limit means that operators must trust one another to act honestly and to not manipulate the consensus protocol.

3.3 Applications

Applications implement the business logic specific to each network. Next to on-chain smart contracts, off-chain applications and data are often required to implement all functionality and integrate with other enterprise systems.

On-chain applications On-chain applications are generally referred to as smart contracts, although this depends on their implementation (see Section 5.7). They promise to replace trust through replicated and verifiable deterministic execution. Since both code and state can be inspected by anyone with access to the ledger, their execution is predictable to these parties. However, a number of smart contract vulnerability studies have shown that code is not always law and may be exploited to an attacker's advantage. For example, in 2016 a symbolic execution tool found almost half of all Ethereum contracts at the time to be vulnerable [40]. In another study, 2 out 5 deployed Ethereum contracts were shown to require trust in at least one third party, since parts of their control flow may be changed after deployment [26].

Off-chain applications. One example for off-chain applications are web applications for user interaction with the distributed ledger. Without a way to verify what is going on behind the scenes, users must blindly trust that the application does not manipulate any data sent through it. While this is also the case for traditional web applications, a distributed system aiming to create trust should provide stronger guarantees.

Table 1. Summary of distributed ledger trust components by layer.

Protocols	Overlay Network	Applications
Storage	Operations	On-chain applications
Cryptography	Identity	Off-chain applications
Network	Consensus	Off-chain data

Off-chain data. Full replication of the blockchain data structure mandates parsimony w.r.t. transaction sizes. Distributed ledger applications rely on off-chain storage solutions to manage larger data volumes. In fact, a recent study found that a majority of DLT operators only include hashes in on-chain transactions [32], which point to off-chain data and serve as integrity timestamps. Operators must trust their peers to maintain sustained availability, since off-chain data is not fully replicated. If off-chain data is access protected, the storage operator must also be trusted to maintain correct access privileges.

Besides referenced data, external data may also be needed as input for computation (i.e. currency exchange rates). Since external data sources must return deterministic results, distributed ledgers rely on trusted external content providers (oracles) - hereby reintroducing trust elements.

A summary of all identified trust components is shown in Table 1. Overall, the complexity of the DLT software layers results in a high degree of obscurity. It becomes increasingly difficult to verify correctness and security of the software stack. The trust actors (operators, users and software service providers) must trust both software components and each other to act as expected. In the next section, we focus on how insiders can exploit these trust assumptions.

4 Insider threats

Given the aforementioned trust elements required for operating a permissioned distributed ledger network, insider attacks may pose a significant threat. With the emergence of business networks and blockchain consortia for data sharing, the partners' information systems infrastructures are no longer isolated environments with a protectable logical perimeter. Access to an organization's resources is implicitly simplified for outsiders that are part of the consortium. This is a direct consequence of sharing information with other organizations.

As a result, a holistic security model must also include actors from network partners. This group of threat actors is also known as **external insiders**. External insiders are characterized by having limited access to an organization's network as a result of some business relationship [25]. In a distributed ledger context, the relationship may be the result of a collaborative network with multiple organizations.

Subsequently, we describe the various insider threats to distributed ledgers by analyzing each group of trust actors. Irrespective of an insider's group, there are generally four types of consequences an insider might achieve in an attack [37]:

Table 2. Overview of insider threats and consequences.

Insider type	Threat	MO	DE	DI	DU
Software service provider	Vulnerability injection	x	x	x	x
Operator	Denial of service				x
	Data manipulation	x	x		x
	Credential compromise	x		x	x
	Malicious misconfiguration			x	x
User	Unauthorized operations	x			x
All	Vulnerability abuse	x	x	x	x
	Information leakage			x	

- Modification (MO)
- Destruction (DE)
- Disclosure (DI)
- Denial of use (DU)

Insiders may exist in any of the three groups of trust actors stated in Figure 1. Depending on the group, there are different ways to exploit their privileges. Table 2 lists the major categories of insider threats and their consequences in distributed ledger consortia. While many of these threats are also applicable to existing information systems, distributed ledgers are particularly vulnerable due to the large number of software components and cross-organizational users.

Permanent modification or destruction of data are generally difficult to achieve with DLT due to built-in fault tolerance and replication. Nevertheless, collusion-based data manipulation or software vulnerabilities may cause data manipulation on all nodes. Disclosure of information and denial of use are the more likely consequences of an insider attack on a distributed ledger. They are significantly easier to accomplish and may be achieved with user or operator level permissions. We elaborate on these threats in detail hereafter.

4.1 Software service providers

If an insider is in the role of an internal software developer with full code access, there is significant threat potential for any of the four consequences to happen. Collins et al. [15] have surveyed a variety of methods that programmers acting as malicious insiders have used in the past. Common methods are code modification or injection of malicious code, causing vulnerabilities. Characteristic for this type of manipulation is a time delay between injection and impact of the attack, since software builds go through testing and deployment phases. In large software projects without strict code review procedures, these types of manipulations may easily go under the radar of other developers. Vulnerabilities can then be abused by the programmer or colluding operators/users. They may corrupt the integrity and availability guarantees that a distributed ledger provides, leading to manipulation or loss of information. Intentionally timed bugs may cause network

unavailability. Backdoors (either for outsiders or insiders) could be inserted that lead to disclosure of confidential information.

The potential attack vectors depend on the developer's area of responsibility (protocol/framework/application level). Currently, distributed ledger protocols and overlay-level frameworks are often provided by open-source initiatives. If an open-source project is subject to peer-review and security audits, these parts of the software are unlikely to be affected by insider attacks threatening a single organization. However, applications built on these frameworks must be customized to specific business requirements — either by employees or third party developers. For this reason, the application level is more prone to software development insider threats. Business networks may collaboratively develop applications, which extends this attack vector to external insiders.

4.2 Operators

Overall operators have the largest number of insider attack options at their disposal, since they directly control a network node. Even though a single operator controls only one node, malicious behavior may have powerful denial-of-service effects on networks with few participants, as mentioned in Section 3.2.

In larger networks, operators could take advantage of their knowledge about the current consensus leader. A malicious operator may launch a targeted denial of service attack to cause network interruptions (see Section 5.6). Additionally, collusion of operators from other organizations (i.e. by external insiders) might result in denial of service, if it leaves the network unable to reach consensus. Generally, network operators have a common goal and should not be inclined to collude against others. This might change if goals shift, or partners feel that they contribute more to the network than they gain in return.

An insider might attempt to accomplish modification of stored data by coordinating an attempt to replace ledger data in collusion with other nodes. This type of attack is known as a 51%-attack for permissionless blockchains [35]. Generally this is only feasible if consensus is stochastic, while permissioned networks usually rely on deterministic consensus. Nevertheless, availability of off-chain data with low replication factors is still at risk.

System administrator insiders have access to all relevant credentials for node operation. These credentials can be leaked, or misused by adding/removing users or manipulating access control privileges at will.

Another way to subvert data integrity is configuration manipulation. Successful configuration manipulation requires collusion, since such changes need to be approved by a majority of operators in properly configured networks. Without automated punishment mechanisms, a single misbehaving node may however still cause temporary service disruption.

4.3 Users

Users have only a limited number of options for exploiting the distributed ledger network. Nevertheless, due to external insiders the number of potential attackers

is higher than in intra-organizational applications. Improperly managed access rights for these users may enable leakage of confidential information.

Another attack vector are vulnerabilities in custom-developed contracts. They could enable insiders to carry out unauthorized asset transfers or even shut down an application. Insiders might have increased knowledge about application internals such as access to source code and technical documentation, enhancing their ability to discover programming flaws and carry out unauthorized operations.

It is important to note that threats are not strictly restricted to a specific group of trust actors. For example, operators may also impersonate users if they control the identity component. Conversely, users may gain operator-level privileges through improper access right management.

In fact, some threats are exploitable by any actor with access to the distributed ledger network. Whether intentional or inadvertent, vulnerabilities can be abused by any insider with the required knowledge and skills. Since all nodes of the network likely run the same software, remote code execution vulnerabilities may lead to irreversible manipulation or loss of data. Similarly, any participant with access to distributed ledger data may leak data to parties outside the network. The extent of information leaked depends on the insider's access privileges.

5 Insider threat analysis of popular frameworks

To demonstrate applicability of the identified insider threats to permissioned blockchain frameworks, we conduct a technical threat analysis of several popular blockchain frameworks. Frameworks were selected based on a survey of industry and research support conducted in July 2019. Regarding industry support, the Ethereum Enterprise Alliance (240 members), the Hyperledger project (249 members) and R3 (92 members and 294 partners) were the largest enterprise consortia working on open-source permissioned DLT frameworks. To gauge research interest, we searched several literature databases for mentions of permissioned distributed ledger frameworks. We used fulltext search since some framework names are ambiguous and might be used with a different meaning in a distributed systems context (i.e. Quorum). The search result counts totaling close to 1000 academic publications are shown in Figure 4. As a result of this survey we decided upon the four frameworks detailed below.

We briefly describe each framework below and summarize the technological components in Table 4. To future-proof our analysis, we mainly analyze threats resulting from architectural design choices, which are unlikely to change in the future. We assume that operators strive for a secure configuration, which includes a byzantine-fault tolerant (BFT) consensus algorithm to prevent byzantine manipulations.

Hyperledger Fabric is a blockchain framework relying on a novel execute-order-validate architecture. This architecture was created to rule out source of

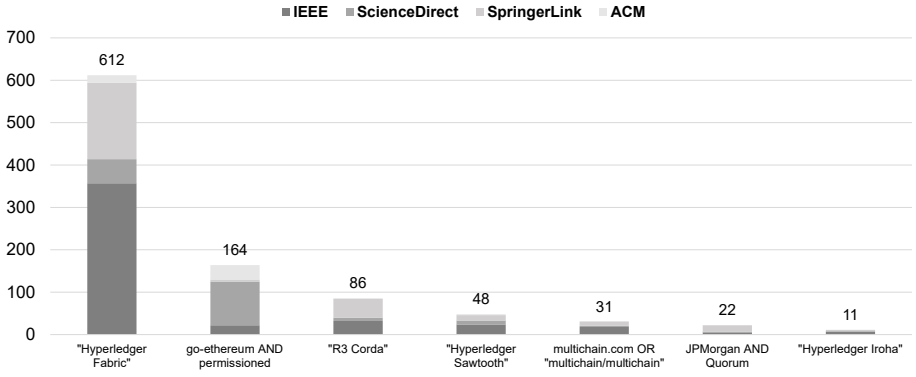

Fig. 4. Research popularity of distributed ledger frameworks (search result count by search term and academic database).

non-determinism during consensus and improve performance [4]. We assume that the BFT-SMaRt consensus algorithm[1] [9] is used for ordering service consensus, since it is to our knowledge the only currently available BFT consensus module for Fabric.

Hyperledger Sawtooth [49] is a blockchain framework, which modularizes transaction processing with so-called transaction families. They include predefined families for permissioning and on-chain settings management. Consensus is also modular, but we assume that the Practical Byzantine Fault Tolerance (PBFT)[2] [12] module is used.

Ethereum [18, 24] is a popular permissionless blockchain framework, which runs smart contracts written in the Solidity language in an isolated environment called the Ethereum Virtual Machine. The go-ethereum client can also be set up as a permissioned network with Clique Proof-of-Authority (PoA) consensus. PoA is a leader-based consensus protocol with stochastic consensus, which only provides eventual consistency as opposed to strong consistency provided by BFT algorithms. It only requires 50% for a consensus majority, trading consistency for availability.

R3 Corda [31, 44] is based on a DAG data structure and only shares data with other nodes when needed. To prevent double spending, mutually agreed upon notary service clusters are used for consensus. We assume that BFT-SMaRt consensus is used among notaries, since it is the only built-in consensus algorithm which tolerates byzantine faults.

All of these frameworks have unique differences in their architecture and the way applications are built on them. While some threats are applicable to all frameworks, others apply only to specific frameworks due to architectural choices. Subsequently, we survey each framework's trust components and analyze where insiders may abuse architectural flaws.

[1] github.com/bft-smart/fabric-orderingservice
[2] github.com/hyperledger/sawtooth-pbft

Table 3. Mapping of insider threats to abused distributed ledger trust components.

Threat	5.1	5.2	5.3	5.4	5.5	5.6	5.7	5.8	5.9
Vulnerability injection		x		x		x	x	x	
Denial of service				x		x			
Data manipulation					x	x			x
Credential compromise					x				
Malicious misconfiguration				x	x	x			
Unauthorized operations							x	x	
Vulnerability abuse	x		x				x	x	
Information leakage	x				x				x

Table 3 highlights which framework components that each type of threat exploits. The columns represent the subsections corresponding to the affected trust components. Corresponding to the trust actors that each threat applies to (Table 2) and their ability to access various trust layers (Figure 3) some cells are marked gray (not applicable). Vulnerability injection does not result in a vulnerability abuse threat if the vulnerability can only be effectively abused by the software service provider. Hereafter, we explain per component how each threat may occur. Threats are italicized when they refer to a table entry.

5.1 Storage

The surveyed distributed ledger frameworks mostly rely on existing key-value databases for data storage (i.e. LevelDB and CouchDB). None of these databases offer encryption-at-rest, which means anyone with access to the database can read all historical data contained in the ledger. Corda marks the exception: it relies on relational databases, some of which offer encryption. Nevertheless, the database itself is an attractive attack vector for operator insiders, who may circumvent framework-level access control by directly accessing the underlying database *(information leakage)*.

5.2 Cryptography

Currently, distributed ledgers almost exclusively use public key cryptography for authentication. The reviewed frameworks use NIST-recommended ECDSA curves in combination with SHA2 for digital signatures, with some also offering EdDSA and RSA. These algorithms are vulnerable to quantum attacks based on Shor's algorithm [6] *(vulnerability abuse)*. Such attacks threaten the authenticity of transactions and network messages (see Section 5.5). If symmetric keys were encrypted using public key cryptography, this may also result in *information leakage*. Once quantum computers reach sufficient computational power, current ledgers will have to be rebuilt from scratch with new identity schemes. Instead of relying on a single cryptographic primitive, developers should instead adopt a

more future-proof approach. For example, quantum-proof hash-based signature schemes use hash combiners, which remain secure if at least one of the input hash functions is secure [6]. In our review, only Corda offers such a scheme with SPHINCS256[3]. Still, no framework provides guidance for migration between signature schemes.

Due to the unique challenges of trust in distributed environments, some distributed ledger frameworks rely on new variants of cryptographic protocols with unproven implementations (especially novel non-interactive zero-knowledge proofs such as zk-STARKs [8]). While we are not aware of any cryptographic flaws in the reviewed permissioned frameworks, there are examples among permissionless blockchains. Recently, a zero-knowledge proof vulnerability in the permissionless blockchain Zcash was publicized, which had been kept secret by the development team for more than 11 months [48]. In this case the developer that discovered the bug did not have malicious intentions and worked on fixing the bug instead of exploiting it. But the incident shows that open-source code is not immune to longstanding hidden vulnerabilities. These may even be inserted into the code intentionally by members of the development team *(vulnerability injection)*. If discovered by malicious actors, they could be kept secret for continued exploitation. Overall, trust in the security of cryptographic protocols is not guaranteed and may be undermined at any time.

5.3 Network protocols

The reviewed frameworks rely on different network protocols for node-to-node communication. The ZeroMQ protocol[4] used in Sawtooth and the AMQP protocol[5] used in Corda have experienced denial of service and remote code execution vulnerabilities in the past[6]. External insiders, who know about the underlying protocols, may abuse these vulnerabilities to cause damage to specific competitors in the network *(vulnerability abuse)*.

Consensus protocols require constant network communication between all involved nodes. For this reason, they are vulnerable to network-partitioning attacks such as Border Gateway Protocol (BGP) hijacking and Eclipse attacks [35]. These attacks have so far mainly been observed and studied on permissionless blockchains. In permissioned blockchains, manipulations of the routing protocol or network traffic interception can also lead to network partitions [22]. If none of the partitions are large enough to reach consensus, the network will stop processing incoming transactions (deterministic algorithms) or create competing forks (stochastic algorithms) [21]. For some consensus algorithms, these network partitions can even allow malicious double-spending transactions (see Section 5.6).

[3] sphincs.cr.yp.to
[4] zeromq.org
[5] www.amqp.org
[6] cve.mitre.org

Table 4. Overview of software components used in popular distributed ledger frameworks

	Hyperledger Fabric v1.4	**Hyperledger Sawtooth v1.1**	**Go-Ethereum v1.9**	**Corda v4.1**
On-chain contracts	Chaincode (Go, nodeJS, Java)	Transaction Processor (see below)	Smart Contract (Solidity)	CorDapps (Java)
Off-chain applications	Go, Java, nodeJS, Python	Go, Java, nodeJS, Python, C++, C#, Swift	Web3 (nodeJS)	Java
Off-chain data	-	-	Swarm	Oracles
Operations	ReST Operations Service, CLI	Settings TP, CLI	RPC CLI	RPC CLI
Identity	Membership Service Provider	Identity Transaction Processor	Accounts	Hierarchical PKI, Doorman Service
Consensus	Endorsements (custom), Ordering (Kafka, BFT-SMaRt)	Journal (PoET, Raft, PBFT)	PoA, IBFT	Notary (Raft, BFT-SMaRt)
Storage	LevelDB, CouchDB	LMDB	LevelDB, RocksDB	H2, Postgres, SQLServer
Cryptography	ZKP: idemix Signature: ECDSA P256/384, Hash: SHA256, SHA3 Encryption: AES	Hash: SHA256/512 Signature: libsecp256k1	Hash: Keccak Signature (using SHA256/384/512, AES): ECDSA P256, P384, P521, S256, bn256	Hash: SHA256 Signature (using SHA256/512, AES): RSA; ECDSA secp256r1, secp256k1; EdDSA-ed25519; SPHINCS256
Network	GRPC, Gossip[a]	ZeroMQ	devP2P[a]	AMQP

[a] custom protocol

5.4 Operations

Regarding operational tools, the frameworks offer little in terms of monitoring capabilities. Only command-line interfaces (CLI) and transaction types for on-chain settings (Hyperledger Fabric and Sawtooth) are offered to retrieve metrics and update settings. Without significant effort by the operators, this may lead to manipulations of configuration settings going undetected *(malicious misconfiguration)*. Additionally, intentional manipulations of consensus network traffic are nearly impossible to detect without proper monitoring. For example, TCP or UDP flooding attacks reduce transaction throughput to a small fraction of peak throughput [14]. Lack of monitoring facilities effectively allows operator (external) insiders to control network throughput by launching attacks when desired *(denial of service)*.

Despite their initial immutability, smart contracts can be upgraded in all surveyed frameworks[7]. If only bytecode is available for inspection, there is no easy way to tell what part of the contract was changed. Since smart contracts may contain vulnerabilities or require feature extension, upgrades cannot be regarded as unusual per default. Individual operator insiders may abuse this fact by upgrading a contract with malicious functionality *(vulnerability injection)*. Requiring signatures of multiple operators for a successful upgrade is a potential mitigation, but the contract needs to be set up with multiple owners for this to apply (Ethereum, Fabric). Corda requires all participants that share a state to sign contract upgrades, but contract signature constraints also permit custom rules that require less signatures for an upgrade [44]. Sawtooth offers no mechanism for coordinated upgrades. Transaction processors run as independent processes next to the validator network and are upgraded by each operator individually.

5.5 Identity and access control

All permissioned networks must admit identities through some form of a gatekeeper. Distributed ledger frameworks attempt to decentralize admission of validating nodes by voting on new members. This does not apply to users, who must receive a certificate through a validating node or its certificate authority. In Ethereum and Sawtooth, there is no certificate infrastructure integration: users either create accounts themselves or request them from a node administrator. However, account pseudonyms need to be mapped to real-world identities for many applications. A lack of certificates complicates permission revocations, which then need to be performed on the application level. Lack of a single source of truth for identity also leads to excessive access rights over time, which enable insider abuse [27]. A potential consequence of unchecked access rights is *information leakage*.

Conversely, Hyperledger Fabric and Corda rely on traditional hierarchical PKIs with a root authority. In these systems, each node operates its own certifi-

[7] Ethereum only allows upgrades if the contract has been set up in a modular way

cate authority. Certificates are then shared across the network through a federation service. As a result, each validator node recognizes the identities issued by its peers. From an insider perspective, this means that any employee with access to the gatekeeper of a participating organization can create valid identities for the entire network. By subverting the local certificate authority, operator insiders may replace associated node's certificate and impersonate it *(credential compromise)*. Accordingly, a collusion between CA operators can subvert multiple node identities and overtake the network, thus enabling *data manipulation*.

Fundamentally, the identity component relies on the underlying cryptographic signature protocols. If a cryptographic primitive is broken, credentials can be forged by issuing fake signatures. Fake signatures in turn enable *data manipulation* and *malicious misconfiguration* through malicious transactions.

5.6 Consensus

The surveyed permissioned distributed ledger frameworks rely on crash fault-tolerant (CFT) or byzantine fault-tolerant (BFT) algorithms. CFT algorithms do not tolerate any malicious activity and are built only to tolerate crashes [10]. As a result they are prone to manipulation by any one operator and not well suited for usage in semi-trusted environments like business networks. Despite this, most frameworks in our survey recommend CFT protocols and mark BFT consensus implementations as experimental.

If operators use BFT algorithms, up to f malicious nodes among $n = 3f + 1$ total nodes are tolerated without ceasing operation. Byzantine failures encompass all possible failure modes of a system. The performance of most BFT algorithms is however heavily impacted by the presence of failures and no consensus is reached with more than f failures. As a result, a single operator can significantly decrease throughput in PBFT-based networks with $n < 7$ independent nodes ($3f + 1 = 7 \mid f = 2$) by flooding the network with messages [14]. Collusion between two operators can even shut down network consensus and prevent new transactions *(malicious misconfiguration)*. Therefore, smaller permissioned networks are especially at risk of *denial of service* by a minority of participants. In addition to flooding-based denial of service, malicious consensus leaders can also degrade performance in most BFT consensus protocols [5].

The PoA algorithm used by permissioned Ethereum networks is vulnerable to the Attack of the Clones [22]. In the attack, a single malicious node can double spend with high probability. By cloning itself and intercepting messages, a network partition is created. The victim partition is deceived by submitting a conflicting transaction to the other partition, which is later accepted as the canonical transaction *(data manipulation)*. Based on the authors' assessment, the only viable countermeasures are switching to a BFT algorithm or requiring a two-thirds majority instead of the current 50%.

In addition to protocol flaws, *vulnerability injection* in the consensus protocol implementation may lead to nodes accepting invalid or malicious transactions. This could be abused by SSP insiders to circumvent on-chain access permissions and transfer assets or tokens.

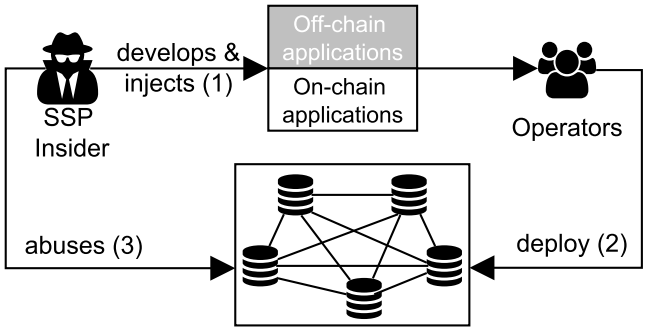

Fig. 5. Illustration of injection and delayed abuse of a vulnerability by a SSP insider.

5.7 On-chain applications

Transparency is an often-cited advantage of smart contracts. In the surveyed frameworks, contract code is rarely transparent to all operators, and never to users. In Hyperledger Fabric, chaincode source code is only known to the peers specified in its endorsement policy. For Sawtooth, bytecode is deployed when using the Seth (Solidity), Sabre and Java SDKs. For transaction processors based on the Python and nodeJS SDKs the source code is transparent, since they are interpreted languages. In Ethereum, only compiled Solidity bytecode is visible to blockchain node operators. Corda's CorDapps are only shared by peers concerned with the application, who need to compute the state changes for notary consensus. A lack of transparency can lead to undetected manipulations, for example through covert contract upgrades.

To demonstrate how this might occur in practice, Figure 5 illustrates the three steps of *vulnerability injection* and subsequent *vulnerability abuse*. First, the SSP insider injects a vulnerability into an on-chain or off-chain application (1). With the next scheduled operational software upgrade, operators deploy the vulnerable version of the software to the production network (2), permitting the insider to abuse the vulnerability (3).

For Hyperledger Fabric, the two most popular SDKs on GitHub are based on nodeJS and Go. Both languages allow package imports from public version control sites such as GitHub. This method could be abused by a software service provider to conceal malicious functionality in the chaincode or insert a backdoor. By changing the code of a self-controlled dependency, the developer gains the ability to manipulate dependent code sections, while obscuring the changes from the client. Additionally, existing vulnerabilities in packages may be knowingly included by a SSP insider to be exploited later on. Due to the large number of transitive dependencies, the nodeJS dependency management system npm is especially prone to attacks based on existing vulnerabilities [19]. For Ethereum smart contracts, many vulnerability classes are known [35] due to public scrutiny and open source application bytecode. For Sawtooth and Corda there are not

many production deployments yet, so to the best of our knowledge we are not aware of any vulnerabilities.

To summarize, chaincode and smart contract vulnerabilities may manipulate the output state of a contract *(data manipulation)*, prevent consensus by introducing non-determinism *(denial of service)*, and leak secrets by sending confidential data to parties outside the network *(information leakage)*. Additionally, users may be able to conduct *unauthorized operations* due to a smart contract permission management vulnerability.

5.8 Off-chain applications

Production deployments of DLT must include a client software, since direct interaction with a blockchain node is not user-friendly and requires command line skills. This client software relies on Software Development Kits (SDKs) published by software service providers. Table 4 shows the diverse programming languages that these SDKs use. For SDKs using package managers, the same attack vector from Section 5.7 applies *(vulnerability injection)*. SSP insiders may abuse SDKs or client software to hijack user identities and thus gain access to the distributed ledger network *(credential compromise)*.

5.9 Off-chain data

For referenced off-chain data, the reduced replication factor exposes data availability to collusion attacks. Depending on the replication factor r of items stored off-chain, r operators may collude to irreversibly delete a key-value pair stored off-chain *(data manipulation)*.

Off-chain events providing external data are not natively supported by the surveyed frameworks. The requirement for code determinism runs counter to uncertain responses from external sources. Consequently, external data is often integrated via trusted applications that attempt to guarantee response integrity. These applications are referred to as validation oracles [52]. They act as automated arbitrators that sign transactions referencing external data on demand. Corda is the only framework that provides built-in integration with centralized oracle services for this purpose. They are accepted as an authoritative source of data by a set of peers. An insider may compromise that service and manipulate transactions at will, without needing access to the distributed ledger *(data manipulation)*. Depending on the application, such attacks can be hard to discover, since the oracle service is not transparent to all operators. Alternative proposals that avoid relying on a centralized provider include secure hardware architectures such as Town Crier [53], and cryptocurrency-based decentralized blockchain oracles such as Astraea [1].

6 Mitigations and future research

The previous sections have shown how various types of trust actors in a distributed ledger system may abuse trust components and carry out insider at-

tacks. Based on these insights, we now elaborate how DLT adopters can better assess which components they trust, and how they can mitigate resulting insider threats.

6.1 A realistic view of trust in distributed ledgers

Instead of regarding blockchain as "trustless", DLT adopters should be aware of the technological components that their trust relies on. First and foremost, software trust management processes should be established to ensure that trust is warranted. The inherent failure modes and consequences should be integrated into organizational risk management.

Regarding trust in the various software components of distributed ledgers, software trust research has established trust principles and an ordered set of classes for software trust measurement [3]. The classes range from Untrusted (T0) to Trusted (T5) and require a progressively larger set of trust principles to be fulfilled. Classes T4 and T5 aim to prevent malicious activity and could be used to certify components for inclusion in trusted distributed ledgers.

To reduce the implicit trust resulting from allowing others to manage identities and access rights, trust-based distributed access control models could be used. Such frameworks include risk assessment processes that dynamically adapt to users' behavior [7]. Additionally, next generation decentralized blockchain-based identity management could enable consensus-based trust in external users, instead of relying on federated membership schemes.

To increase trust in smart contracts, a number of approaches have been proposed. Business and legal smart contract specification languages based on formal reasoning can help reduce ambiguity for programmers and lock down edge cases [2].

To make user interaction with distributed ledgers more transparent, "decentralized applications" (DApps) may be used. DApps are web applications relying solely on an on-chain application backend. DApp frontends should be distributed as client-only applications, with code only served from, but not executed on a centralized web server. This ensures that code execution is fully transparent to end-users. Transaction submission to the blockchain can be explicitly authorized using open source browser extensions (i.e. MetaMask [42]).

6.2 Insider threat mitigation

Techniques for insider threat mitigation have been studied extensively in the past [16]. They require an interdisciplinary mitigation approach, combining insights from computer science, psychology and other fields. Correspondingly, mitigation techniques can be classified as technical or organizational. We have analyzed mitigation techniques in the literature and applied them to the threats mentioned in Table 2. The result of this analysis is shown in Table 5. We emphasize technical measures, but organizational mechanisms are sometimes required and often beneficial.

Table 5. Overview of potential mitigations for insider threats. O: organizational, T: technical.

	Threat	**Mitigation**	**O/T**
Software service provider	Vulnerability injection	Software trust management	O
		Code review and transparency	O,T
		Vulnerability scanners	T
Operator	Denial of service	Legal agreements	O
		Robust consensus algorithms	T
	Data manipulation	Automated detection and punishment	T
	Credential compromise	Credential revocation mechanisms	T
	Malicious misconfiguration	Configuration integrity checks	T
User	Unauthorized operations	Activity monitoring	O,T
		Anomaly detection	T
All	Information leakage	Granular Identity and Access Management	T
		Granular encryption	T
	Vulnerability abuse	Software update management	T

To counter injection of vulnerabilities or flaws by **software service providers**, *code review* processes help assure sufficient oversight and reduce the probability for malicious activity to go unnoticed. From a framework perspective, *transparency* should be assured by embedding on-chain application code into the distributed ledger framework. Participants should be able to retrieve source code for a contract at any given time. Re-compiling source-code from a repository is too time-consuming and error-prone to serve as a verification method for smart contract bytecode.

Additionally, dedicated *vulnerability scanners* exist specifically for distributed ledger on-chain applications [40]. Whether these scanners are suitable for detecting insider-induced intentional manipulations of program flow has yet to be shown. Future empirical research may also analyze specific programming techniques that insiders use to attack distributed ledgers, similar to the study conducted by Collins et al. [15].

Intentional *denial of service* caused by a collusion of **operators** may be mitigated through legal agreements and incentives. While research on collusion in distributed ledger networks is still scarce, trust-based reputation algorithms could help. They may prevent collusion attempts or at least minimize their impact on network availability by punishing colluding peers. In the past, reputation systems based on peer-to-peer networks have also faced the issue of collusion [41]. Future research could thus transfer insights on collusion prevention from peer-to-peer reputation systems research and related areas to distributed ledgers.

Regarding consensus attacks targeting network availability and throughput, *robust consensus algorithms* such as RBFT [5] can help. Robust protocols sacrifice some throughput compared to traditional algorithms [14], but maintain high availability and throughput regardless of attacks.

Activity monitoring tools and corresponding organizational processes assist with timely detection of data manipulation. The threats listed in this work can serve as guidance for activities to monitor. To prevent insider abuse by unchecked node administrators, monitoring should be part of IS security management and

organizationally separated from operational IS administration. Similarly, certificate authority and DLT node should be managed by separate entities. If any entity attempts to manipulate node settings, automated *configuration integrity checks* should raise alerts in security monitoring units across the network. In case of external insiders, attack attempts should be punished either through legal agreements or a built-in incentive system.

Transaction *anomaly detection* could help spot unauthorized operations. If anomalies are detected in time, further abuse of permissions can be prevented. For traditional database systems, tools have been developed that automatically determine profiles of normal activity based on application profiles [34]. Similar security analytics tools can be developed to detect anomalous smart contract transactions.

One of the main countermeasures for information leakage by **users** is *granular identity and access management* (IAM) [27]. If users cannot send transactions or access ledger data without first being authorized by an application owner, the attack surface becomes significantly smaller. For authorized users, automated detection and timely removal of access privileges helps limit permission buildup and impact of insider attacks. Still, it remains challenging to ensure that each operator of the network keeps its individual permissions and access rights updated. Future work could examine how to enforce granular IAM network-wide, for example through automated checks or incentives.

Another tool to limit malevolent information disclosure is *granular encryption* of data, ensuring that users are only able to view data they need to access. Organizations must correctly decide where to encrypt data on the application level. Additionally, they should be parsimonious regarding confidential data stored on the ledger. Symmetric encryption keys may eventually be leaked, but on-ledger data cannot be deleted.

Overall, we observe that a holistic security monitoring concept is necessary for each organization participating in a distributed ledger network. A set of standardized monitoring metrics is a necessary prerequisite to detect manipulations of the various trust components. Due to unique differences in distributed ledger architectures, the metrics need to be customizable to the specific application context. One metric should be vulnerabilities in framework components, with automated *software update processes* attempting to minimize the number of vulnerabilities. Future work should focus on creating and evaluating such a security management framework for permissioned distributed ledger networks.

7 Conclusion

While distributed technology offers great benefits, organizations planning to take advantage of it should be aware of the trust relationships they enter. In this work, we established key trust actors and components for distributed ledgers to provide a better understanding of hidden trust factors and security risks. On the one hand, software trust in the components of a distributed ledger system is a key factor. If there are vulnerabilities or bugs, trust assumptions may be

violated with grave consequences. Additionally, operators must still trust one another to some degree to cooperatively run a network. If this trust turns sour during operation, the distributed ledger network may become subject to denial of service or collusion attacks.

These attacks may be especially severe if carried out by insiders. They possess unique access to organizational resources that may facilitate subversion of distributed ledger trust assumptions. Insiders involved in application development may willingly inject vulnerabilities or malicious code. Node administrator insiders have elevated access rights to credentials and configuration. Malicious manipulation of these components may result in denial of service for the entire network. Modification or destruction of data are also possible in some cases (see Section 4), despite the integrity guarantees that distributed ledgers normally provide. Both internal and external insiders may leak data or abuse vulnerabilities in the distributed ledger software stack.

Due to the current lack of productive deployments of distributed ledger networks, this work focused on analyzing the potential impact of insider attacks from a theoretical perspective. To reinforce these claims, we elaborated on how insiders may exploit the architecture of popular distributed ledger frameworks. Future research may conduct case studies with real deployments of these frameworks to validate the listed insider threats and to further develop mitigations.

References

1. Adler, J., Berryhill, R., Veneris, A., Poulos, Z., Veira, N., Kastania, A.: Astraea: A decentralized blockchain oracle. 2018 IEEE International Conference on Blockchain (2018). https://doi.org/10.1109/Cybermatics_2018.2018.00207
2. Al Khalil, F., Butler, T., O'Brien, L., Ceci, M.: Trust in smart contracts is a process, as well. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2017). https://doi.org/10.1007/978-3-319-70278-0_32
3. Amoroso, E., Nguyen, T., Weiss, J., Watson, J., Lapiska, P., Starr, T.: Toward an approach to measuring software trust. In: Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy. pp. 198–218 (1991). https://doi.org/10.1109/RISP.1991.130788
4. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S.W., Yellick, J.: Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In: Proceedings of the Thirteenth EuroSys Conference. pp. 30:1–30:15. EuroSys '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3190508.3190538
5. Aublin, P.L., Mokhtar, S.B., Quema, V.: RBFT: Redundant byzantine fault tolerance. Proceedings - International Conference on Distributed Computing Systems pp. 297–306 (2013). https://doi.org/10.1109/ICDCS.2013.53
6. Bansarkhani, R.E., Geihs, M., Buchmann, J.: PQChain: Strategic design decisions for distributed ledger technologies against future threats. IEEE Security and Privacy (2018). https://doi.org/10.1109/MSP.2018.3111246

7. Baracaldo, N., Joshi, J.: A Trust-and-Risk Aware RBAC Framework : Tackling Insider Threat. In: SACMAT '12:Proceedings of the 17th ACM symposium on Access Control Models and Technologies (2012)
8. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Eprint.Iacr.Org (2018). https://doi.org/10.1016/j.bspc.2009.02.004
9. Bessani, A., Sousa, J., Alchieri, E.E.P.: State Machine Replication for the Masses with BFT-SMART. Dsn **6897**(December), 355–362 (2014). https://doi.org/10.1109/DSN.2014.43
10. Cachin, C., Vukolic, M.: Blockchain Consensus Protocols in the Wild. In: Richa, A.W. (ed.) 31st International Symposium on Distributed Computing (DISC 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 91, pp. 1:1—-1:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017). https://doi.org/10.4230/LIPIcs.DISC.2017.1
11. del Castillo, M.: Blockchain 50: Billion Dollar Babies (2019), `https://www.forbes.com/sites/michaeldelcastillo/2019/04/16/blockchain-50-billion-dollar-babies`
12. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems **20**(4), 398–461 (2002). https://doi.org/10.1145/571637.571640
13. Chia, V., Hartel, P., Hum, Q., Ma, S., Piliouras, G., Reijsbergen, D., van Staalduinen, M., Szalachowski, P., Staalduinen, M.V., Szalachowski, P.: Rethinking Blockchain Security: Position Paper. 2018 IEEE International Conference on Blockchain (2018), `http://arxiv.org/abs/1806.04358`
14. Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M.: Making Byzantine fault tolerant systems tolerate Byzantine faults. In: NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation (2009). https://doi.org/10.1145/1989727.1989732
15. Collins, M., Cappelli, D.M., Caron, T., Trzeciak, R.F., Moore, A.P.: Spotlight On: Programmers as Malicious Insiders–Updated and Revised. Tech. rep., Software Engineering Institute, Carnegie Mellon University (2013)
16. Colwill, C.: Human factors in information security: The insider threat - Who can you trust these days? Information Security Technical Report (2009). https://doi.org/10.1016/j.istr.2010.04.004
17. Dasgupta, D., Shrein, J.M., Gupta, K.D.: A survey of blockchain from security perspective. Journal of Banking and Financial Technology (jan 2019). https://doi.org/10.1007/s42786-018-00002-6
18. De Angelis, S., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A., Sassone, V.: PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain. CEUR Workshop Proceedings **2058**, 1–11 (2018)
19. Decan, A., Mens, T., Grosjean, P.: An empirical comparison of dependency network evolution in seven software packaging ecosystems. Empirical Software Engineering (2019). https://doi.org/10.1007/s10664-017-9589-y
20. Deventer, M.O., Berkers, F., Vos, M., Zandee, A., Vreuls, T., van Piggelen, L., Blom, A., Heeringa, B., Akdim, S., van Helvoort, P., Others: Techruption Consortium Blockchain: what it takes to run a blockchain together. In: Proceedings of 1st ERCIM Blockchain Workshop 2018, Amsterdam, Netherlands 8-9 may 2018. European Society for Socially Embedded Technologies (EUSSET) (2018)
21. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.L.: BLOCKBENCH: A Framework for Analyzing Private Blockchains. In: Proceedings of the 2017

ACM International Conference on Management of Data. pp. 1085–1100. SIGMOD '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3035918.3064033, http://doi.acm.org/10.1145/3035918.3064033
22. Ekparinya, P., Gramoli, V., Jourjon, G.: The Attack of the Clones against Proof-of-Authority. CoRR (2019), http://arxiv.org/abs/1902.10244
23. ENISA: ENISA Threat Landscape Report 2018. Tech. rep., ENISA (2019). https://doi.org/10.2824/622757
24. Ethereum Foundation: Go-Ethereum Website (2019), https://geth.ethereum.org
25. Franqueira, V.N.L., v. Cleeff, A., v. Eck, P., Wieringa, R.: External Insider Threat: A Real Security Challenge in Enterprise Value Webs. In: 2010 International Conference on Availability, Reliability and Security. pp. 446–453 (feb 2010). https://doi.org/10.1109/ARES.2010.40
26. Fröwis, M., Böhme, R.: In Code We Trust? In: Garcia-Alfaro, J., Navarro-Arribas, G., Hartenstein, H., Herrera-Joancomartí, J. (eds.) Data Privacy Management, Cryptocurrencies and Blockchain Technology. pp. 357–372. Springer International Publishing, Cham (2017)
27. Fuchs, L., Pernul, G.: Minimizing insider misuse through secure Identity Management. Security and Communication Networks (2012). https://doi.org/10.1002/sec.314
28. Gambetta, D.: Can We Trust Trust? In: Trust: Making and Breaking Cooperative Relations, pp. 213–237. Blackwell (1988)
29. Glaser, F.: Pervasive Decentralisation of Digital Infrastructures: A Framework for Blockchain enabled System and Use Case Analysis. HICSS 2017 Proceedings pp. 1543–1552 (2017). https://doi.org/10.1145/1235
30. Hawlitschek, F., Notheisen, B., Teubner, T.: The limits of trust-free systems: A literature review on blockchain technology and trust in the sharing economy. Electronic Commerce Research and Applications **29** (2018). https://doi.org/10.1016/j.elerap.2018.03.005
31. Hearn, M.: Corda: A distributed ledger (2016), https://docs.corda.net/head/static/corda-technical-whitepaper.pdf
32. Hileman, G., Rauchs, M.: 2017 Global Blockchain Benchmarking Study (2017)
33. Huseby, D.: Security Code Audits - Hyperledger Wiki (2019), https://wiki.hyperledger.org/display/HYP/Security+Code+Audits
34. Hussain, S.R., Sallam, A., Bertino, E.: DetAnom: Detecting anomalous database transactions by insiders. In: CODASPY 2015 - Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (2015). https://doi.org/10.1145/2699026.2699111
35. Li, X., Jiang, P., Chen, T., Luo, X., Wen, Q.: A survey on the security of blockchain systems. Future Generation Computer Systems (2017). https://doi.org/https://doi.org/10.1016/j.future.2017.08.020, http://www.sciencedirect.com/science/article/pii/S0167739X17318332
36. Litke, A., Anagnostopoulos, D., Varvarigou, T.: Blockchains for Supply Chain Management: Architectural Elements and Challenges Towards a Global Scale Deployment. Logistics **3**(1) (2019). https://doi.org/10.3390/logistics3010005
37. Loch, K.D., Carr, H.H., Warkentin, M.E.: Threats to Information Systems: Today's Reality, Yesterday's Understanding. MIS Quarterly (1992). https://doi.org/10.1163/18781527-00401005
38. Locher, T., Obermeier, S., Pignolet, Y.A.: When Can a Distributed Ledger Replace a Trusted Third Party? 2018 IEEE International Conference on Blockchain (2018), http://arxiv.org/abs/1806.10929

39. Lustig, C., Nardi, B.: Algorithmic authority: The case of Bitcoin. In: Proceedings of the Annual Hawaii International Conference on System Sciences (2015). https://doi.org/10.1109/HICSS.2015.95
40. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making Smart Contracts Smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16 (2016). https://doi.org/10.1145/2976749.2978309
41. Marti, S., Garcia-Molina, H.: Taxonomy of trust: Categorizing P2P reputation systems. Computer Networks (2006). https://doi.org/10.1016/j.comnet.2005.07.011
42. MetaMask Contributors: MetaMask (2019), https://metamask.io/
43. Muskens, J., Chaudron, M.: Integrity management in component based systems. In: Proceedings. 30th Euromicro Conference, 2004. pp. 611–619 (2004). https://doi.org/10.1109/EURMIC.2004.1333429
44. R3: Corda Documentation (2019), https://docs.corda.net/releases/release-V4.1/
45. Sas, C., Khairuddin, I.E.: Exploring Trust in Bitcoin Technology: A Framework for HCI Research. Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction on - OzCHI '15 (2015). https://doi.org/10.1145/2838739.2838821
46. Schaffers, H.: The Relevance of Blockchain for Collaborative Networked Organizations. In: Camarinha-Matos, L.M., Afsarmanesh, H., Rezgui, Y. (eds.) Collaborative Networks of Cognitive Systems. pp. 3–17. Springer International Publishing, Cham (2018)
47. Schneier, B.: Blockchain and Trust - Schneier on Security (2019), https://www.schneier.com/blog/archives/2019/02/blockchain_and.html
48. Swihart, J., Winston, B., Bowe, S.: Zcash Counterfeiting Vulnerability Successfully Remediated - Zcash (2019), https://z.cash/blog/zcash-counterfeiting-vulnerability-successfully-remediated/
49. The Linux Foundation: Hyperledger Sawtooth Documentation (2019), https://sawtooth.hyperledger.org/docs/core/releases/1.1.5/contents.html
50. United States Department of Homeland Security: A Roadmap for Cybersecurity Research (2009). https://doi.org/10.1016/j.biortech.2007.06.061
51. Vo, H.T., Wang, Z., Karunamoorthy, D., Wagner, J., Abebe, E., Mohania, M.: Internet of Blockchains: Techniques and Challenges Ahead. In: 2018 IEEE International Conference on Blockchain. IEEE (2018). https://doi.org/10.1109/Cybermatics_2018.2018.00264
52. Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A.B., Chen, S.: The blockchain as a software connector. In: Proceedings - 2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016. pp. 182–191. IEEE (apr 2016). https://doi.org/10.1109/WICSA.2016.21
53. Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E.: Town Crier: An Authenticated Data Feed for Smart Contracts. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (2016). https://doi.org/10.1145/2976749.2978326